Search History

Database Details

Set	Term Searched	Items	
S1	SOFTWARE(N)METRIC?	464	Display
S2	S1 AND INDEX?	22	Display
S3	RULE(W)BASED OR PREDEFINED(W)RULE?	1553	Display
S4	S1 (S) S3	1	Display
S5	SOFTWARE(N)(METRIC? OR MEASUREMENT?)	567	Display
S6	S3 AND S5 AND INDEX? AND PY<1996	3	Display
S7	S3 (S) INDEX?	22	Display
S8	S5 AND S7	0	Display
S9	S5(S)S3	1	Display
S10	S5 AND S3 AND PY<1996	5	Display
S11	S5 AND FORMAT?(N)CONVERSION?	1	Display
S12	SCHEMA(W)TUNNEL?	0	Display
S13	S5 (S) UNIVERSAL?	1	Display
S14	(DATA)(W)(DEFINITION? OR COMPONENT?)	908	Display
S15	S5 (S) S14	0	Display
S16	S5 AND S14 AND PY <1996	0	Display
S17	S5 (S) SOFTWARE?	566	Display
S18	S5 (S) SOFTWARE? (S) (METRIC?)	466	Display
S19	S18 AND PY<1996	452	Display
S20	S19 AND INDEX?	22	Display

Format
Free ▼

#Documents
10

Show Database Details for:

275: IAC(SM) Computer Database(TM) ▼   Bluesheet   Rates

Fields   Formats   Sorts   Limits   Tags

DATE ISSUED: Jun. 16, 1992
TITLE: Method and system of retrieving program specification and
                 linking the specification by concept to retrieval
                 request for reusing program parts
INVENTOR: Noriko Ohtaki, Machida, Japan
                 Yoshiaki Nagai, Yokohama, Japan
                 Yuji Magamatsu, Kawasaki, Japan
                 Eiki Chigira, Tokyo, Japan
ASSIGNEE: Hitachi, Ltd., Tokyo, Japan (foreign corp.)
APPL-NO: 07/109,269
DATE FILED: Oct. 15, 1987
FRN-PRIOR: Japan   61-246791   Oct. 17, 1986
                 Japan   62-33232   Feb. 18, 1987
INT-CL: :5: G06F 15/40
US-CL-ISSUED: 395/600; 364/282.1, 283.4, 274, 286, DIG.1
US-CL-CURRENT: 707/5; 364/222.81, 222.82, 225, 225.1, 234, 237.2, 274,
                 274.1, 274.2, 274.8, 282.1, 282.4, 283.3, 283.4, 286,
                 DIG.1; 395/703, **710**
SEARCH-FLD: 364/200MSFile, 900MSFile
REF-CITED:

	U.S. PATENT DOCUMENTS		
4,283,771	8/1981	Chang	364/200
4,384,329	5/1983	Rosenbaum	364/300
4,455,619	6/1984	Masui	364/900
4,497,039	1/1985	Kitakami	364/200
4,498,142	2/1985	Advani	364/900
4,506,326	3/1985	Shaw	364/900
4,555,771	11/1985	Hayashi	364/900
4,558,413	12/1985	Schmidt	364/200
4,611,298	9/1986	Schuldt	364/900
4,631,664	12/1986	Bachman	364/200
4,636,974	1/1987	Griffin	364/900
4,644,471	2/1987	Kojima	364/300
4,674,066	6/1987	Kucera	364/900
4,680,705	7/1987	Shu	364/300
4,714,995	12/1987	Materna	364/200
4,734,854	3/1988	Afshar	364/200
4,742,467	5/1988	Messerich	364/200
4,769,772	9/1988	Dwyer	364/200
4,774,661	9/1988	Kumpati	364/900
4,809,170	2/1989	Leblang	364/200
4,819,156	4/1989	DeLorme et al.	364/200
4,825,354	4/1989	Agrawal	364/200
4,827,411	5/1989	Arrowood et al.	364/200
4,829,427	5/1989	Green	364/200
4,833,601	5/1989	Cheng	364/200
4,835,683	5/1989	Phillips	364/200
4,864,497	9/1989	Lowry	364/200
4,891,441	6/1989	Nixon	364/300
4,894,771	1/1990	Kunii	364/300

ART-UNIT: 232
PRIM-EXMR: Thomas C. Lee
ASST-EXMR: Eric Coleman
LEGAL-REP: Fay, Sharpe, Beall, Fagan, Minnich & McKee

ABSTRACT:
Herein discolosed is a software information reusing system comprising: a

data base for storing software information; a dialogue display terminal; thereby processing the information inputted from the display terminal. First retrieval information of the software information is registered as data related with a link of a data information related concept. If second retrieval information is not in the registered data, it is judged on the basis of the link of the related concept whether or not third retrieval information related therewith exists. The data base is retrieved on the basis of the third retrieval information. As a result, it is possible: to form a dictionary data base including a noun dictionary and a synonym dictionary for analyzing and generalizing the data object of the specification to be newly registered; for a user unacquainted with the business knowledge to automatically extract a proper retrieval keyword from a retrieval request sentence describing the software coming into the mind of the user; and to judge from the relations between the keyword information and the before and behind words whether or not even a composed word left either unregistered as one word or unknown is the retrieval keyword, if even one of elements composing the composed word is found to have the corresponding keyword information.

17 Claims, 22 Drawing Figures

US PAT NO: 5,123,103 :IMAGE AVAILABLE: L3: 11 of 12

CLAIMS:

=> d 13 1,5,11

1. 5,659,751, Aug. 19, 1997, Apparatus and method for dynamic linking of
computer software components; Andrew G. Heninger, 395/685, 683, **710**
:IMAGE AVAILABLE:

5. 5,553,290, Sep. 3, 1996, Software packaging structure having
hierarchical replaceable units; Nathaniel Calvert, et al., 395/703;
364/280, 286, DIG.1; **395/710** :IMAGE AVAILABLE:

11. 5,123,103, Jun. 16, 1992, Method and system of retrieving program
specification and linking the specification by concept to retrieval
request for reusing program parts; Noriko Ohtaki, et al., 707/5;
364/222.81, 222.82, 225, 225.1, 234, 237.2, 274, 274.1, 274.2, 274.8,
282.1, 282.4, 283.3, 283.4, 286, DIG.1; 395/703, **710** :IMAGE
AVAILABLE:

US PAT NO:	5,261,100 :IMAGE AVAILABLE:	
DATE ISSUED:	Nov. 9, 1993	
TITLE:	Method of software development	
INVENTOR:	Tsutomu Fujinami, Kawasaki, Japan	
	Hirohide Haga, Kyoto, Japan	
ASSIGNEE:	Hitachi, Ltd., Tokyo, Japan (foreign corp.)	
APPL-NO:	07/363,509	
DATE FILED:	Jun. 8, 1989	
FRN-PRIOR:	Japan 63-144526	Jun. 10, 1988
INT-CL:	:5: G06F 15/40	
US-CL-ISSUED:	395/700, 650, 919, 922; 364/DIG.1, 275.1	
US-CL-CURRENT:	395/703; 364/225.6, 225.8, 234, 236.8, 237.2, 237.3, 238.3, 243, 243.3, 259.4, 262, 265, 274, 274.2, 274.3, 275.1, 286, 286.1, DIG.1; 395/706, 919, 922	
SEARCH-FLD:	364/200, 900; 395/600, 650, 700, 919, 922	
REF-CITED:		

U.S. PATENT DOCUMENTS

4,330,822	5/1982	Dodson	364/200
4,734,854	3/1988	Afshar	364/200
4,809,170	2/1989	Leblang et al.	364/200
4,827,404	5/1989	Barstow et al.	364/200
4,833,641	5/1989	Lerner	364/900
4,860,204	8/1989	Gendron et al.	364/300
4,949,253	8/1990	Chigira et al.	364/200
4,956,773	9/1990	Saito et al.	364/200
4,974,160	11/1990	Bone et al.	364/200
5,005,119	4/1991	Rumbaugh et al.	364/200
5,084,813	1/1992	Ono	395/1
5,101,491	3/1992	Katzeff	395/500
5,123,103	6/1992	Ohtaki et al.	395/600

OTHER PUBLICATIONS

Shi-Kuo Chang, IEEE Software, vol. 4, pp. 29-39, Jan. 1987.
Interactive Programming Environments, Barstow et al. 1984 pp. 370-413.
Komiya, et al., "Automatic Programming by Fabrication of Reusable Program Components", Information Processing, vol. 28, No. 10, 1987, pp. 1329-1345.

ART-UNIT:	237
PRIM-EXMR:	David L. Clark
ASST-EXMR:	John Loomis
LEGAL-REP:	Fay, Sharpe, Beall, Fagan, Minnich & McKee

ABSTRACT:

A program data managing apparatus comprising memories for storing as program data a source code, technique data on a process for making the source code, and intention data on intention to make the source code; a link indicative of the mutual relationship between program data; a display for displaying the relationship between the program data using the link; a link provided to indicate the relationship between a newly developed source code and the original program data from which the new source code derives; and a display for displaying the source code developed stepwise by that link and the related program data.
18 Claims, 34 Drawing Figures

```
=> d his full


          (FILE 'USPAT' ENTERED AT 12:40:21 ON 15 JUN 1998)
L1          64 SEA SOFTWARE#(A)(METRIC# OR MEASUREMENT#)
L2        1066 SEA 395/70?,71?/CCLS
L3           5 SEA L1 AND L2
L4           5 SEA L1 AND HIERARCH?
L5          24 SEA L1 AND INDEX?
L6           2 SEA L4 AND L5
L7          10 SEA L1 AND (INDEX?)/TI,AB,CLM
L8           1 SEA L1 AND INSTANTIAT?
L9          26 SEA L1 AND INSTANCE#
L10         27 SEA (HIERARCH?)(A)(INDEX#### OR INDICES)
L11          0 SEA L1 AND L10
L12          0 SEA L10 (P) SOFTWARE?
L13         13 SEA L10 AND SOFTWARE?
```

1. **5,699,310**, Dec. 16, 1997, Method and apparatus for a fully inherited object-oriented computer system for generating source code from user-entered specifications; Gary W. Garloff, et al., 395/701 :IMAGE AVAILABLE:

2. **5,606,661**, Feb. 25, 1997, Apparatus and method for scan-based testing in an object-oriented programming environment; Larry L. Wear, et al., 395/183.14, 704; 707/103 :IMAGE AVAILABLE:

3. **5,526,522**, Jun. 11, 1996, Automatic program generating system using recursive conversion of a program specification into syntactic tree format and using design knowledge base; Hiroshi Takeuchi, 395/702; 364/274.1, 274.3, 274.5, 280.4, 972.3, 973, DIG.1, DIG.2; 395/10, 922 :IMAGE AVAILABLE:

4. **5,261,100**, Nov. 9, 1993, Method of software development; Tsutomu Fujinami, et al., 395/703; 364/225.6, 225.8, 234, 236.8, 237.2, 237.3, 238.3, 243, 243.3, 259.4, 262, 265, 274, 274.2, 274.3, 275.1, 286, 286.1, DIG.1; 395/706, 919, 922 :IMAGE AVAILABLE:

5. **5,159,687**, Oct. 27, 1992, Method and apparatus for generating program code files; Joseph B. Richburg, 395/702, 50, 703, 922 :IMAGE AVAILABLE:

6. **5,123,103**, Jun. 16, 1992, Method and system of retrieving program specification and linking the specification by concept to retrieval request for reusing program parts; Noriko Ohtaki, et al., 707/5; 364/222.81, 222.82, 225, 225.1, 234, 237.2, 274, 274.1, 274.2, 274.8, 282.1, 282.4, 283.3, 283.4, 286, DIG.1; 395/703, 710 :IMAGE AVAILABLE:

```
        (FILE 'USPAT' ENTERED AT 13:01:36 ON 14 JAN 1998)
L1         871 S 395/70?,71?/CCLS
L2          12 S ((INDEX?)(P)(HIERARCH?)) AND L1
L3         264 S SOFTWARE(A)(METRIC# OR MEASUR? OR TESTING)
L4          18 S L1 AND L3

=> d l4 4

4.. 5,606,661, Feb. 25, 1997, Apparatus and method for scan-based testing
in an object-oriented programming environment; Larry L. Wear, et al.,
395/183.14, 704; 707/103 :IMAGE AVAILABLE:
```

Search Options:
    Search for both singular and plurals: YES
    Search for spelling variants   : YES
    Display intermediate result sets   : NO

Num	Search	Hits
#1	software? W/2 (metric? OR measur?)	717
#2	#1 AND (tree? OR hierarch?)	31
#3	#1 AND (rule-based OR expert OR knowledge?)	44

```
=> d his

      (FILE 'USPAT' ENTERED AT 13:01:36 ON 14 JAN 1998)
L1          871 S 395/70?,71?/CCLS
L2           12 S ((INDEX?)(P)(HIERARCH?)) AND L1
L3          264 S SOFTWARE(A)(METRIC# OR MEASUR? OR TESTING)
L4           18 S L1 AND L3
L5            3 S (5129083 OR 5255385 OR 5485617)/PN
L6            0 S 5606661/UREF,BI
L7         2394 S (INSTANCE# OR INSTANTIAT?)(2A)(OBJECT# OR INDEX?)
L8          289 S L7 (20A)(CHNAGE# OR LINK#### OR DIFFEREN? OR RELATIONSHI
P#)
L9           27 S L1 AND L8
L10           2 S L3 AND L9
L11           9 S (MEASUREMENT)(W)(CLASS##)
```

Display Sets

Search History

Database Details

Set	Term Searched	Items	
S1	FRAMEWORK (2W) SOFTWARE	438	Display
S2	FRAMEWORK (5W) SOFTWARE (5W) MEASUREMENT?	0	Display
S3	SOFTWARE (5W) MEASUREMENT?	330	Display
S4	S3 (S) FRAMEWORK?	9	Display

Format

Free ▼

#Documents

10

Show Database Details for:

275: IAC(SM) Computer Database(TM) ▼   Bluesheet   Rates

Fields   Formats   Sorts   Limits   Tags

© 1997 Knight-Ridder Information, Inc.

Search Options:
    Search for both singular and plurals: YES
    Search for spelling variants   : YES
    Display intermediate result sets   : NO

Num   Search   Hits
#1   software W/2 measurement?   224
#2   #1 AND validation?   10

```
Set      Items    Description
S1        166     AMADEUS
S2        100     S1 AND SOFTWARE?
S3          2     S2 AND METRIC?
S4        934     (SOFTWARE?)(S)(METRIC OR METRICS)
S5        114     (SOFTWARE?)(S)(METRIC OR METRICS)(S)(MEASUREMENT?)
```

```
Set       Items    Description
S1          567    (SOFTWARE?)(N)(MEASUREMENT? OR METRIC?)
S2            6    S1 (S) INDEX?
S3         5208    RULE(W)BASED OR (KNOWLEDGE(W)(BASE? OR PROCESS?))
S4           34    S1 AND S3
S5            7    S4 AND INDEX?
S6           34    S4 AND PY <1996
S7           85    SOFTWARE(W)MEASUREMENT?
S8            2    S7 (S) (RULE? OR KNOWLEDGE? OR EXPERT(W)SYSTEM?)
```

```
=> d his

        (FILE 'USPAT' ENTERED AT 08:46:09 ON 14 JAN 1998)
L1           59 S SOFTWARE#(A)(METRIC# OR MEASUREMENT#)
L2           23 S L1 AND INDEX?
L3            1 S L2 AND (RULE(W)BASED OR EXPERT(W)SYSTEM# OR (KNOWLEDGE?(
W)(
L4            3 S L1 (P) (INDEX?)
L5           43 S 4751635/UREF,BI
L6         2311 S RULE(W)BASED OR EXPERT(W)SYSTEM# OR (KNOWLEDGE?(W)(BASE
OR
L7            4 S L5 AND L6
L8        30715 S 395/CLAS OR 707/CLAS OR 1/1/CCLS OR 705/CLAS
L9           10 S L1 AND L8
L10        1633 S (DATA)(2A)(DEFINITION#)
L11           1 S L1 AND L10
L12         466 S L6 (P) SOFTWARE#
L13          18 S L6 (P) SOFTWARE# (P)(MEASUREMENT# OR METRIC#)
L14          98 S (SOFTWARE#)(A)(RELIABLIT? OR MEASUREMENT# OR PRODUCTION#
  OR
L15           2 S ((SOFTWARE#)(A)(RELIABLIT? OR MEASUREMENT# OR PRODUCTION
# O
L16           6 S L6 AND L14
L17         138 S 707/10/CCLS
L18           2 S L14 AND L17
L19           1 S 5274806/UREF,BI
L20           2 S L14 (P) (HETEROGENEOUS? OR COMPATIB? OR CONVERSION#)
L21          23 S 5159687/UREF,BI
L22           9 S L21 AND INDEX?
L23           3 S L6 AND L22
L24        3532 S SOFTWARE#(A)APPLICATION#
L25         205 S (PLURALITY OR MULTIPLE OR SEVERAL OR DIFFERENT)(5A)L24
L26           9 S L25 (P) (HETEROGENEOUS? OR COMPATIB? OR CONVERSION#)
L27           1 S 5708828/PN
L28           1 S L27 AND GENERIC
L29           1 S L27 AND INDEX?
L30           1 S L27 AND (TREE# OR NODE# OR HIERARCH?)
L31           0 S L27 AND (INDEX?(P)HIERARCH?)
L32         103 S L6 (P) INDEX?
L33          11 S L6 (P) INDEX? (P)(DATA) (P)(DEFINITION# OR COMPONENT# OR
  PA
```

01516949   SUPPLIER NUMBER: 12202262   (THIS IS THE FULL TEXT)
Go with the flow. (combining expert systems and flowcharts to develop
   knowledge bases) (Expert's Toolbox) (Tutorial)
Knaus, Rodger
AI Expert, v7, n6, p17(3)
June, 1992

TEXT:
     Flowcharts and expert systems-what could seem farther apart?
Flowcharts remind us of punched cards and Fortran, and seem too rigid for
the relatively unstructured knowledge behind an expert system. On the other
hand, if you do have a flowchart, why waste time with expert systems
instead of writing a program from the flowchart? Sometimes, however, this
unlikely combination is useful because you can use flowcharts for knowledge
acquisition, and expert-system shells and languages are an easy way to code
flowcharts into runable software. Also, a flow-chart-based **knowledge
base** is easy to change.
     KNOWLEDGE ACQUISITION
     Let's start with knowledge acquisition. As a knowledge engineer, your
goal is to extract problem-solving knowledge from the expert for a
computerized **knowledge base**. Sometimes the experts have already
organized the knowledge into a flowchart for their own use. This was the
case for a patient-management expert system I wrote recently; an
almost-complete flowchart was available in a technical bulletin for
doctors.
     Suppose a flowchart isn't available. For patient management and many
other potential expert-system applications, the problem solver, whether
human expert or expert system, gathers information and makes a
determination using that information. Each problem situation is a dialog
between the expert and client. As a knowledge engineer, you interview the
expert, asking questions "What do you do first?"; "What information do you
look for now?"; "What do you do now?"; "What are the possible situations
that can occur next?"; "What are the conditions under which these
situations occur?" Using these questions, we can lead the expert through
one scenario, then go back to a choice point and take another branch,
continuing the process until all choices are explored.
     As we conduct the interview in this form, we record each dialog as a
path in an emerging flowchart, adding actions, branches, and edges to the
flowchart as needed with each successive dialog. The flowchart neatly
summarizes the information so that the expert can inspect it for
completeness. In addition, each phase of the dialog consists of reporting
actions and decisions in a sequence similar to how they occur in practice;
this makes the form in which knowledge is reported similar to the form in
which it is used.
     ENCODING FLOWCHARTS
     That a flowchart existed for the patient-management problem was the
good part; the bad part was that it grew to contain more than 60 boxes.
Compared to recommended **software metrics**, many expert-provided
flowcharts are too big or contain loops that don't map easily into
structured code. Tasks performed by organizations, such as adding a new
book to a library, also have complex flowcharts, especially when we try to
capture procedures in practice rather than as formally defined.
     While it's always possible to transform a flowchart into one
containing only if-then-else branches, begin-end blocks, and while loops,
this restructuring has several disadvantages: As descriptions of what
should be done, the restructured flowcharts are often harder for the expert

to understand. This makes it harder for the expert to contribute to the development and maintenance of the system. Also, you make mistakes transforming the original flowchart into structured code. Finally, small changes to the original flowchart can require major changes to the restructured one. For these reasons, we want to encode the expert's flowchart in a straightforward, recognizable way into an expert system. Prolog and **rule-based** shells are ideal for this task.

To demonstrate some techniques for turning flowcharts into rules, we'll use the example in Figure 1, which is illustrative only. If we create one rule for each path from the top of the flowchart to the conclusion, we get the rule set shown in Listing 1. The if part of each rule contains the logical conditions that must be satisfied to follow a particular path to a conclusion box; that conclusion is in the then part of the rule.

As the path length from the top to the bottom of the flowchart increases, this logic-based approach becomes increasingly unwieldy, because the hypothesis contains one condition for each edge in the path. In addition, the number of paths flowing through an interior region of the flowchart increases, so a change in the flowchart affects a large number of rules.

We can overcome this problem by using a state variable. Its value, called the current state, tells us where in the flowchart we are at the moment. To represent the current state, we'll assign a unique name to each box in the flowchart and use these names as possible values of the state variable; in Figure 1, the box names appear just outside the box. Listing 2 shows rules for the flowchart rewritten using state variables. Now the complexity of each rule does not change, no matter how long or looping the paths through the flowchart become-although the chance for unexpected computational paths does increase as the flowchart becomes messier.

Now suppose that the expert finds that the original flowchart is not quite right. Each change to a path in the flowchart causes only one rule to change in the **knowledge base,** when using the state variables in the rules. In contrast, if we used conditions on paths, many rules would change. Likewise, there is no single, simple change in a structured program that mirrors a one-edge change in an underlying messy flowchart. Encoding the expert's original flowchart using state variables minimizes the work in responding to changes requested by the expert. It also helps experts understand how the expert system corresponds to their own knowledge.

Listing 3 shows Listing 1 in Prolog. The process predicate holds the expert system rules, and the other predicates run the rules. The when predicate tries to prove that its question argument has the answer specified in its second argument; consider moves the system to a new state; and conclude makes a recommendation to the user. A good strategy for when is backward chaining, with asking the user as a backup source of information; for the implementation details see "A Portable Engine," AI Expert, jan. 1990. The resulting system uses forward chaining at the top level to execute a specified, overall problem-solving strategy, while leaving the details to be figured out by backward chaining. Global forward chaining with local backward chaining seems to represent a good compromise between the unpredictability of pure backward chaining and the detailed programming required for pure forward chaining; some shells, such as Exsys, offer`this hybrid strategy as an option.

The basic structure of the Prolog flowchart program also handles multiple answer questions, such as "Which of the Ten Commandments has the candidate violated?" If we let when return a list of answers and make consider put each corresponding next state in the database, the program will eventually get around to each branch we chose.

As the flowchart gets bigger, your expert system might slow down. This happens if all the process rules are stored in a single linear list searched from the top each time a state is processed. Some Prologs like Quintus) **index** clauses on the first argument, automatically eliminating this problem. If your Prolog doesn't do this, you can program the technique yourself (Listing 4). First reprogram initialize to record the body of each process clause under the state in its head. Then modify process to try each rule of the current state in turn.

HELPING THE EXPERT

By looking at the rules derived from our flowchart, we see that a
knowledge base of this type is really just a table in which
each row represents one rule that corresponds to one line in the flowchart.
An individual row is fully specified when we provide:
* The current state that activates the rule
* Tests that have to be satisfied for the rule to apply
* Actions to be performed once the test is satisfied
* The new current state after performing the actions
* Whether additional rules can apply at the current state

Knowing that this is all we need for the top level of our expert
system, we can say to the expert: First list all the situations or
problem-solving steps that occur when you work on problems that the expert
system should do (column I of the table). Then for each step you listed,
list the questions you would ask and the answers you might receive (column
2). Now, for each partially filled-in row, complete the table. To complete
a **knowledge base**, we would also have to ask an expert for
conditions that imply the tests in the table so the finished system can do
local backward chaining.

I hypothesize that filling in such a table is relatively easy for
experts compared to writing if-then rules with complete logical conditions
for tests and actions. This is because experts usually have years of
practice, and they can replay in their minds what they do step by step.
They also know and can identify situations where tests and actions ought to
be considered, which is what we ask them to do in filling in the table.

On the other hand, writing complete logical conditions for rules is
like explaining expertise to a novice; for both a novice and a computer,
you can't assume that they will be able to fill in obvious details you
leave out. By asking for information in a form that allows experts to
replay their experience, and by coding with states the experts provide, we
can construct, at least for interview-and-decide expert systems, a
knowledge base that any domain expert can understand.

Rodger Knaus is a principal of instant Recall, a Bethesda, Md. firm
specializing in the development of Prolog-based applications.

Listing 1. Rules using logical conditions.
If: Initial test is {negative} then: No further evaluation is needed
If: Initial test is {positive} and: Biopsy contains {no abnormality}
then: Monitor again in 6 months
If: Initial test is {positive} and: Biopsy contains {abnormal cells}
then: Evaluate treatment options

Listing 2. Rules using state variables.
If: {STATE} = "Initial test" and: Initial test is {negative} then: No
further evaluation is needed
If: {STATE} = "Initial test"
If: Initial test is {positive} then: [STATE] := "Perform biopsy"
If: {STATE} = "Perform biopsy" and: Biopsy contains {abnormal cells}
then: {STATE} := Evaluate treatment options
If: {STATE} := "Perform biopsy" and: Biopsy contains {no abnormality}
then: Monitor again in 6 months

Listing 3. Rules using state variables

```
machine :
-       initialize,
        repeat,
            single_state.
single_state :
-       retract( current_state( STATE)), !,
        process( STATE ), fail.
single_state :
-       message($Computation concluded.$).
consider( STATE) :
-       asserta( current_state( STATE)).
initialize:
-       consider( $Initial test$ ).
process( $Initial test$ ) :
-       when( $Initial test is$,$negative$), !,
        conclude( $No further evaluation is needed$ ).
```

```
process( $Initial test$ ) :
-      when( $Initial test$, $positive$), !,
       consider( $Perform biopsy$ ).
process( $Perform biopsy$ ) :
-      when( $Biopsy content$, $no abnormality$ ), !.
       conclude( $Monitor again in 6 months$ ).
process( $Perform biopsy$ ) :
-      when( $Biopsy content$, $abnormal cells$ ), !,
       consider( $Evaluate treatment options$ ).
```

Listing 4. **Indexing** rules by state.

```
initialize
       consider(  Initial test$ ).
       clause( process( STATE) , BODY ),
       recordz( STATE, BODYfail. initialize :- !.
       process( STATE ) :
-       recorded( STATE, RULE_BODY, _ ),
       call( RULE_BODY ), !.
```

5/3,AB/4

01516949   SUPPLIER NUMBER: 12202262   (USE FORMAT 7 OR 9 FOR FULL TEXT)
Go with the flow. (combining expert systems and flowcharts to develop
   knowledge bases) (Expert's Toolbox) (Tutorial)
Knaus, Rodger
AI Expert, v7, n6, p17(3)
June, 1992
DOCUMENT TYPE: Tutorial   ISSN: 0888-3785   LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 1859   LINE COUNT: 00155

ABSTRACT: The unlikely combination of expert systems and flowcharts can be
useful for developing **knowledge bases** such as expert system
shells and languages. While expert systems can be used to gather the
necessary information to solve a particular problem, flowcharts help to
summarize the data so that experts can inspect it for thoroughness. The
expert's flowchart should be encoded using PROLOG and **knowledge-
based** shells. The fundamental structure of the PROLOG flowchart
program handles multiple answer questions. A **knowledge base**
developed using expert systems and flowcharts is essentially a table with
individual rows representing rules that match with individual lines on the
flowchart. By requesting data in a form that enables experts to replay
their experience, and by coding with states the experts offer, expert
system developers can build a **knowledge base** that all domain

01325067   SUPPLIER NUMBER: 08805222
Evaluating techniques for generating **metric**-based classification
   trees. (technical)
Porter, Adam A.; Selby, Richard W.
Journal of Systems and Software, v12, n3, p209(10)
July, 1990
DOCUMENT TYPE: technical   ISSN: 0164-1212   LANGUAGE: ENGLISH
RECORD TYPE: ABSTRACT

ABSTRACT: **Metric**-based classification trees provide an approach for
identifying user-specified classes of high-risk **software** components
throughout the **software** lifecycle. Based on measurable attributes of
software components and processors, this empirically guided approach
derives models of problematic **software** components. These models,
which are represented as classification trees, are used on future systems
to identify components likely to share the same high-risk properties.
Example high-risk component properties include being fault-prone,
change-prone, or effort-prone, or containing certain types of faults.
Identifying these components allows developers to focus the application of
specialized techniques and tools for analyzing, testing, and constructing
software. A validation study using **metric** data from 16 NASA
systems showed that the trees had an average classification accuracy of
79.3 percent for fault-prone and effort-prone components in that
environment. One fundamental feature of the classification tree generation
algorithm is the method used for partitioning the **metric** data values
into mutually exclusive and exhaustive ranges. This study compares the
accuracy and the complexity of trees resulting from five techniques for
partitioning **metric** data values. The techniques are quartiles,
octiles, and three methods based on least weight subsequence (LWS-chi)
analysis, where chi is the upper bound on the number of partitions. The
LWS-3 and LWS-5 partition techniques resulted in trees with higher accuracy
(in terms of completeness and consistency) than did quartiles and octiles.
LWS-3 and LWS-5 trees were not statistically different in terms of
accuracy, but LWS-3 trees had lower complexity than all other methods in
terms of the number of unique **metrics** required. The trees from the
three LWS methods (LWS-3, LWS-5, and LWS-8) had lower complexity than did
the trees from quartiles and octiles. In general, the results indicate that
distribution-sensitive partition techniques that use only relatively few
partitions, such as the least weight subsequence techniques LWS-3 and
LWS-5, can increase accuracy and decrease complexity in classification
trees. Classification analysis techniques, along with other empirically
based analysis techniques for large-scale **software**, will be supported
in the **Amadeus** measurement and empirical analysis system. (Reprinted
by permission of the publisher.)